Agentic System

Tips and Tricks

John Tan Chong Min

Agent Definition

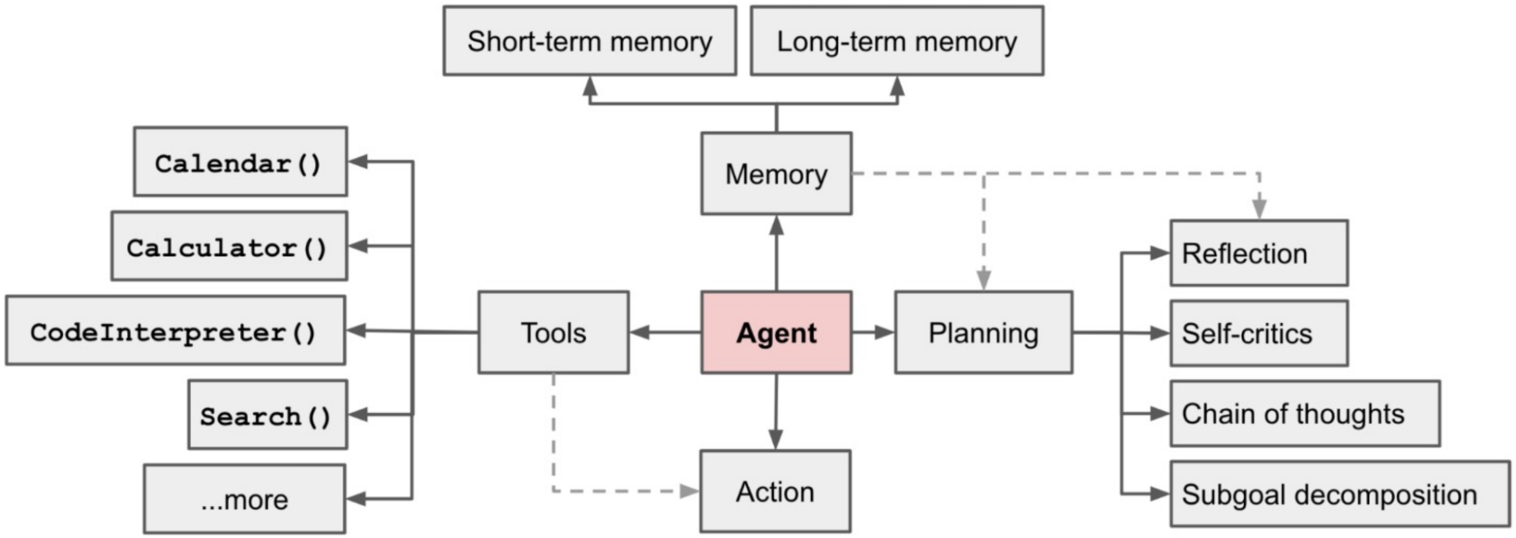
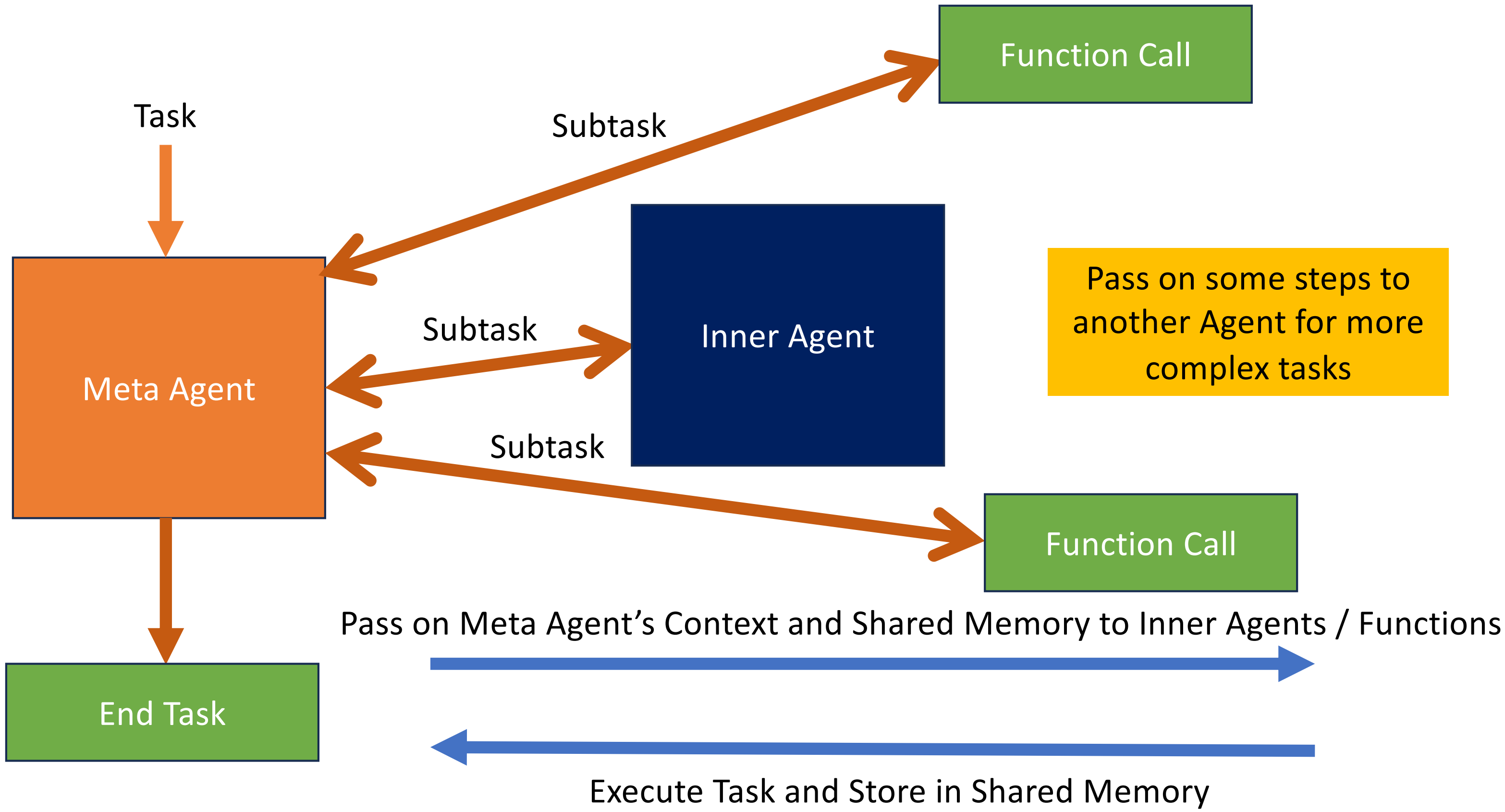


Fig. 1. Overview of a LLM-powered autonomous agent system.

https://lilianweng.github.io/posts/2023-06-23-agent/

TaskGen Overview

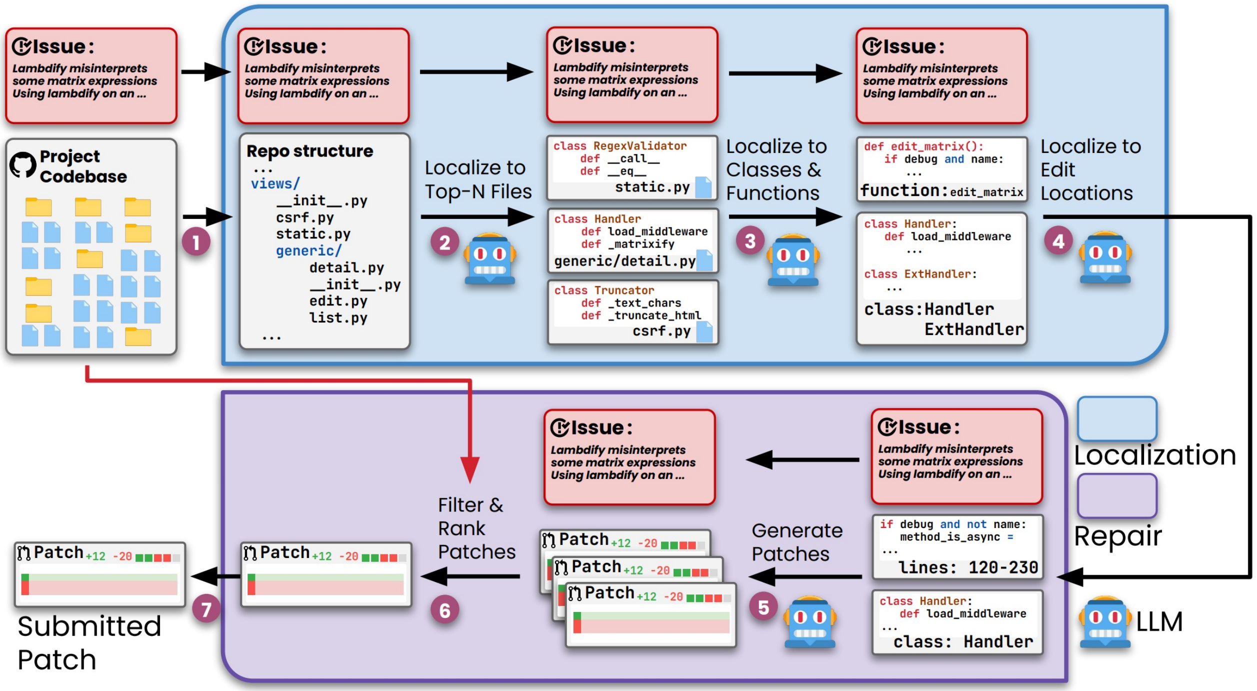


https://github.com/simbianai/taskgen

Process Flow

Fixed Processes help increase reliability

AGENTless



Agentless: Demystifying LLM-based Software Engineering Agents. Chunqiu et al. 2024.

AGENTless Overview

- AGENTLESS does not allow LLMs to autonomously decide future actions or operate with any complex tools

- AGENTLESS fixes the process, then leverages LLMs to perform each detailed task

AGENTLess is competitive with close-sourced systems

Table 4: Performance and ranking on SWE-bench Lite-*S*. * indicates a tie in ranking.

Tool	LLM	SWE-bench Lite % Resolved	SWE-bench Lite Rank	SWE-bench Lite-*S* % Resolved	SWE-bench Lite-*S* Rank
Alibaba Lingma Agent [7] 🔒	GPT-4o+ Claude-3.5	99 (33.00%)	1	87 (34.52%)	1
Factory Code Droid [5] 🔒	NA	94 (31.33%)	2	82 (32.54%)	2
AutoCodeRover-v2 [3] 🔒	GPT-4o	92 (30.67%)	3	79 (31.35%)	3
CodeR [17] 🔒	GPT-4	85 (28.33%)	4	72 (28.57%)	4
IBM Research Agent-101 [6] 🔒	NA	80 (26.67%)	6	66 (26.19%)	7
OpenCSG StarShip [9] 🔒	GPT-4	71 (23.67%)	9	57 (22.62%)	9
Bytedance MarsCode [8] 🔒	GPT-4o	76 (25.33%)	8	63 (25.00%)	8
Amazon Q Developer [1] 🔒	NA	61 (20.33%)	11	52 (20.63%)	10*
RepoUnderstander [41] 🔒	GPT-4	64 (21.33%)	10	52 (20.63%)	10*
Aider [21]	GPT-4o+ Claude-3	79 (26.33%)	7	67 (26.59%)	6
AutoCodeRover [65]	GPT-4	57 (19.00%)	12	46 (18.25%)	12
SWE-agent [61]	Claude-3	35 (11.67%)	16	27 (10.71%)	16
	GPT-4	54 (18.00%)	13	42 (16.67%)	14
OpenDevin [10]	GPT-4	50 (16.67%)	15	41 (16.27%)	15
	GPT-4o	52 (17.33%)	14	45 (17.86%)	13
RAG [28]	Claude-3	13 (4.33%)	17	10 (3.97%)	17
	GPT-4	8 (2.67%)	19	5 (1.98%)	19
	Claude-2	9 (3.00%)	18	6 (2.38%)	18
	ChatGPT	1 (0.33%)	20	0 (0.00%)	20
AGENTLESS 🐱	GPT-4o	82 (27.33%)	5	71 (28.17%)	5

Letting Agent Decide Everything

```
agent = Agent('General Agent', 'Does anything', llm = llm)
```

```
agent.run('Write me a four sentence poem about cars, and extract out all car brands used')
```

```
Observation: No subtasks have been completed yet, and the task of writing a four-sentence poem about cars and extracting car brands is still pendin
g.
Thoughts: To complete the assigned task, I need to generate a four-sentence poem that includes various car brands, and then identify and extract tho
se brands from the poem.
Subtask identified: Write a four-sentence poem about cars that includes multiple car brands, ensuring that the brands are easily identifiable for ex
traction.
Getting LLM to perform the following task: Write a four-sentence poem about cars that includes multiple car brands, ensuring that the brands are eas
ily identifiable for extraction.
> In the city where the Teslas glide,
And the Ford Mustangs roar with pride,
A sleek BMW races down the lane,
While a classic Chevy brings back the old refrain.

Observation: A four-sentence poem about cars has been successfully written, featuring identifiable car brands such as Tesla, Ford, BMW, and Chevy.
Thoughts: The next step is to extract the car brands mentioned in the poem to fulfill the remainder of the Assigned Task.
Subtask identified: Extract the car brands from the poem provided, which includes Tesla, Ford, BMW, and Chevy.
Getting LLM to perform the following task: Extract the car brands from the poem provided, which includes Tesla, Ford, BMW, and Chevy.
> The car brands extracted from the poem are Tesla, Ford, BMW, and Chevy. These brands are clearly identifiable within the context of the poem, as t
hey are mentioned in relation to their characteristics and the imagery of cars in a city setting. Tesla is associated with modernity and innovation,
Ford with power and pride, BMW with speed and luxury, and Chevy with nostalgia and classic appeal. This extraction highlights the diversity of car b
rands and their cultural significance in automotive discussions.
```

Fixing Process, using LLM for each part

```
res = strict_json(system_prompt = 'Generate a four sentence poem about cars, including numerous car brands',
                  user_prompt = '',
                  output_format = {'Poem': 'type: str'},
                  llm = llm)
```

```
poem = res['Poem']
print(poem)
```

```
In a Tesla’s hum, the future gleams bright,\
While a Ford Mustang roars into the night.\
A sleek BMW glides, with elegance and grace,\
As a rugged Jeep conquers every wild place.
```

```
res = strict_json(system_prompt = 'Extract out all car brands present in the poem',
                  user_prompt = poem,
                  output_format = {'Car Brands': 'type: list'},
                  llm = llm)
```

```
brands = res['Car Brands']
print(brands)
```

Conditional Flow along with fixed process

```
res = strict_json(system_prompt = 'Extract out all car brands present in the poem',
                  user_prompt = poem,
                  output_format = {'Car Brands': 'type: list'},
                  llm = llm)
```

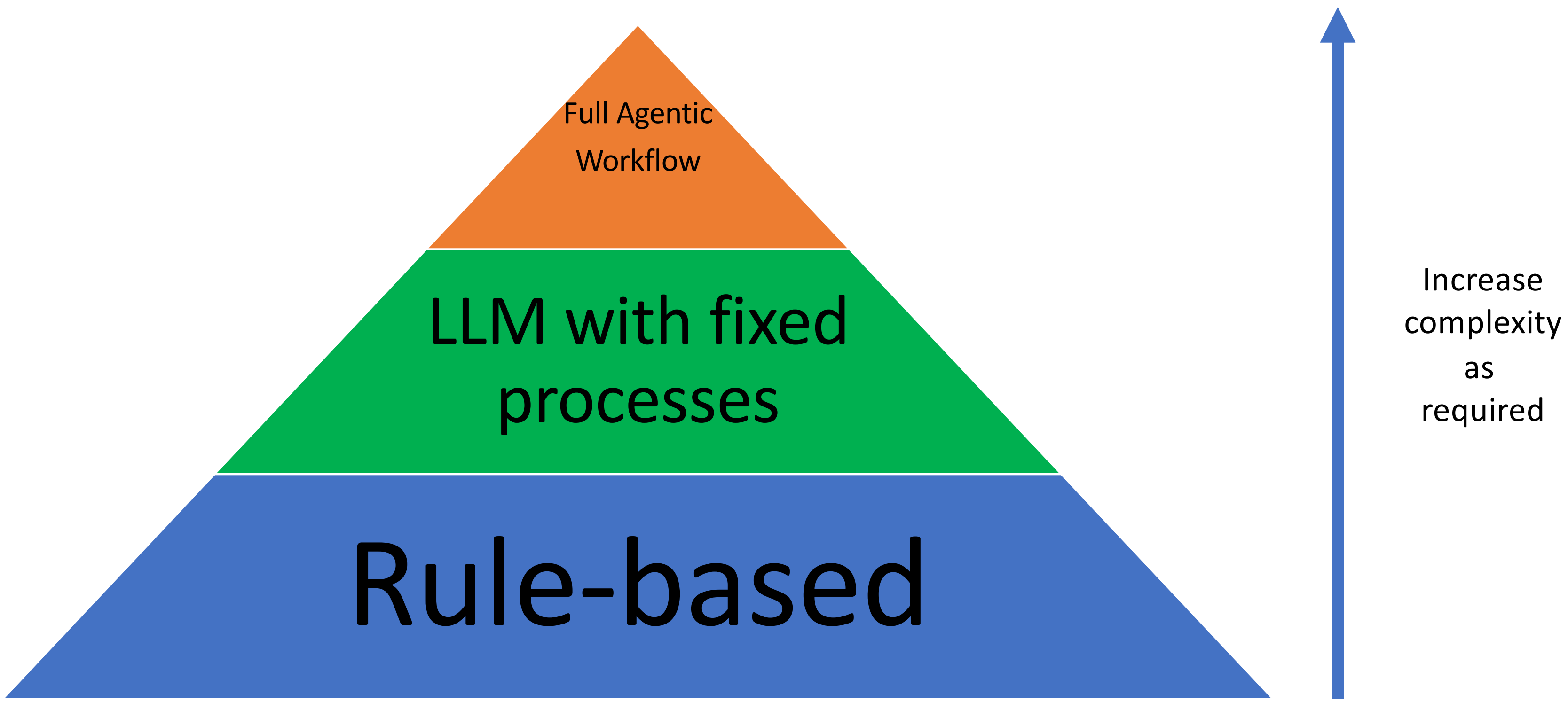
```
brands = res['Car Brands']
print(brands)
```

```
['Tesla', 'Ford', 'Mustang', 'BMW', 'Jeep']
```

```
if 'Tesla' in brands:
    print('You should get a Tesla now')
```

```
You should get a Tesla now
```

Pyramid of Complexity



Tool Use

Fewer tools can be more reliable

Giving more tools is not always good

- Divide and conquer is better approach

- Fewer tools for each agent = better performance

Using more tools is all right if disambiguated well

```
def buy_food(food: str) -> str:
    ''' Buys the food '''
    return 'Food bought'
def buy_drink(drink: str) -> str:
    ''' Buys the drink '''
    return 'Drink bought'
def buy_shirt(shirt: str) -> str:
    ''' Buys the shirt '''
    return 'Shirt bought'
```

```
agent = Agent('General Agent', 'Does anything', llm = llm).assign_functions([buy_food, buy_drink, buy_shirt])
```

```
agent.run('Get me a watermelon juice')
```

```
Observation: No subtasks have been completed yet for the task of getting a watermelon juice.
Thoughts: To complete the task, I need to buy a drink specifically identified as watermelon juice.
Subtask identified: Use the buy_drink function to purchase watermelon juice.
Calling function buy_drink with parameters {'drink': 'watermelon juice'}
> {'output_1': 'Drink bought'}
```

Tools can interfere

```python
def buy_food(food: str) -> str:
    ''' Buys the food '''
    return 'Food bought'
def buy_drink(drink: str) -> str:
    ''' Buys the drink '''
    return 'Drink bought'
def buy_watermelon(amount: int) -> str:
    ''' Buys amount watermelons '''
    return 'Watermelons bought'
def buy_shirt(shirt: str) -> str:
    ''' Buys the shirt '''
    return 'Shirt bought'
```

```python
agent = Agent('General Agent', 'Does anything', llm = llm).assign_functions([buy_food, buy_drink, buy_shirt, buy_watermelon])
```

```python
agent.run('Get me a watermelon juice')
```

```
Observation: No subtasks have been completed yet for the task of getting watermelon juice.
Thoughts: To complete the task of getting watermelon juice, I need to first buy a watermelon, as it is the primary ingredient for the juice.
Subtask identified: Buy 1 watermelon to use for making the juice.
Calling function buy_watermelon with parameters {'amount': 1}
> {'output_1': 'Watermelons bought'}

Observation: One watermelon has been bought as part of the task to get watermelon juice.
Thoughts: Now that I have a watermelon, I need to process it to make watermelon juice. However, there is no specific function for making juice, so I will need to use the general function to handle this.
Subtask identified: Use the general function to process the watermelon into juice.
Getting LLM to perform the following task: Use the general function to process the watermelon into juice.
> The watermelon has been processed into juice using a general function designed for fruit extraction. The watermelon was first washed thoroughly to remove any dirt or pesticides. It was then cut into smaller pieces to facilitate blending. The pieces were placed into a high-powered blender, where they were blended until smooth. After blending, the mixture was strained through a fine mesh sieve to separate the juice from the pulp, resulting in a refreshing watermelon juice. The juice was then chilled and is now ready to be served, providing a delicious and hydrating beverage.
```

Divide and Conquer by Inner Agents (Part 1)

```python
def buy_food(food: str) -> str:
    ''' Buys the food '''
    return 'Food bought'
def buy_drink(drink: str) -> str:
    ''' Buys the drink '''
    return 'Drink bought'
def buy_watermelon(amount: int) -> str:
    ''' Buys amount watermelons '''
    return 'Watermelons bought'
def buy_shirt(shirt: str) -> str:
    ''' Buys the shirt '''
    return 'Shirt bought'
```

```python
agent1 = Agent('Chef', 'Makes dishes', llm = llm).assign_functions([buy_watermelon, buy_food])
agent2 = Agent('Drink Retailer', 'Sells drinks', llm = llm).assign_functions([buy_drink])
agent3 = Agent('Shirt Retailer', 'Sells shirts', llm = llm).assign_functions([buy_shirt])

meta_agent = Agent('General Agent', 'Does anything', llm = llm).assign_functions([agent1, agent2, agent3])
```

Divide and Conquer by Inner Agents (Part 2)

```
meta_agent.run('Get me a watermelon juice')
```

```
Observation: No subtasks have been completed yet for the task of getting a watermelon juice.
Thoughts: To complete the task, I need to execute a function that can sell drinks, specifically watermelon juice.
Subtask identified: Execute the Drink Retailer function to sell watermelon juice.
Calling function Drink Retailer with parameters {'instruction': 'Execute the Drink Retailer function to sell watermelon juice.'}

### Start of Inner Agent: Drink Retailer ###
Observation: No subtasks have been completed yet for the task of selling watermelon juice.
Thoughts: To complete the task of selling watermelon juice, I need to execute the function that allows me to buy the drink. Since the task specifies selling watermelon juice, I will focus on purchasing it first.
Subtask identified: Buy watermelon juice to proceed with the selling process.
Calling function buy_drink with parameters {'drink': 'watermelon juice'}
> {'output_1': 'Drink bought'}

Observation: The drink watermelon juice has been successfully bought.
Thoughts: Since the drink has been purchased, the next step is to finalize the transaction and provide confirmation to the user.
Subtask identified: End Task
Task completed successfully!

###
Reply from Drink Retailer to General Agent:
The Drink Retailer function has been executed successfully to sell watermelon juice. The subtasks completed indicate that the drink has been bought, confirming that the transaction for watermelon juice is complete.
###
```

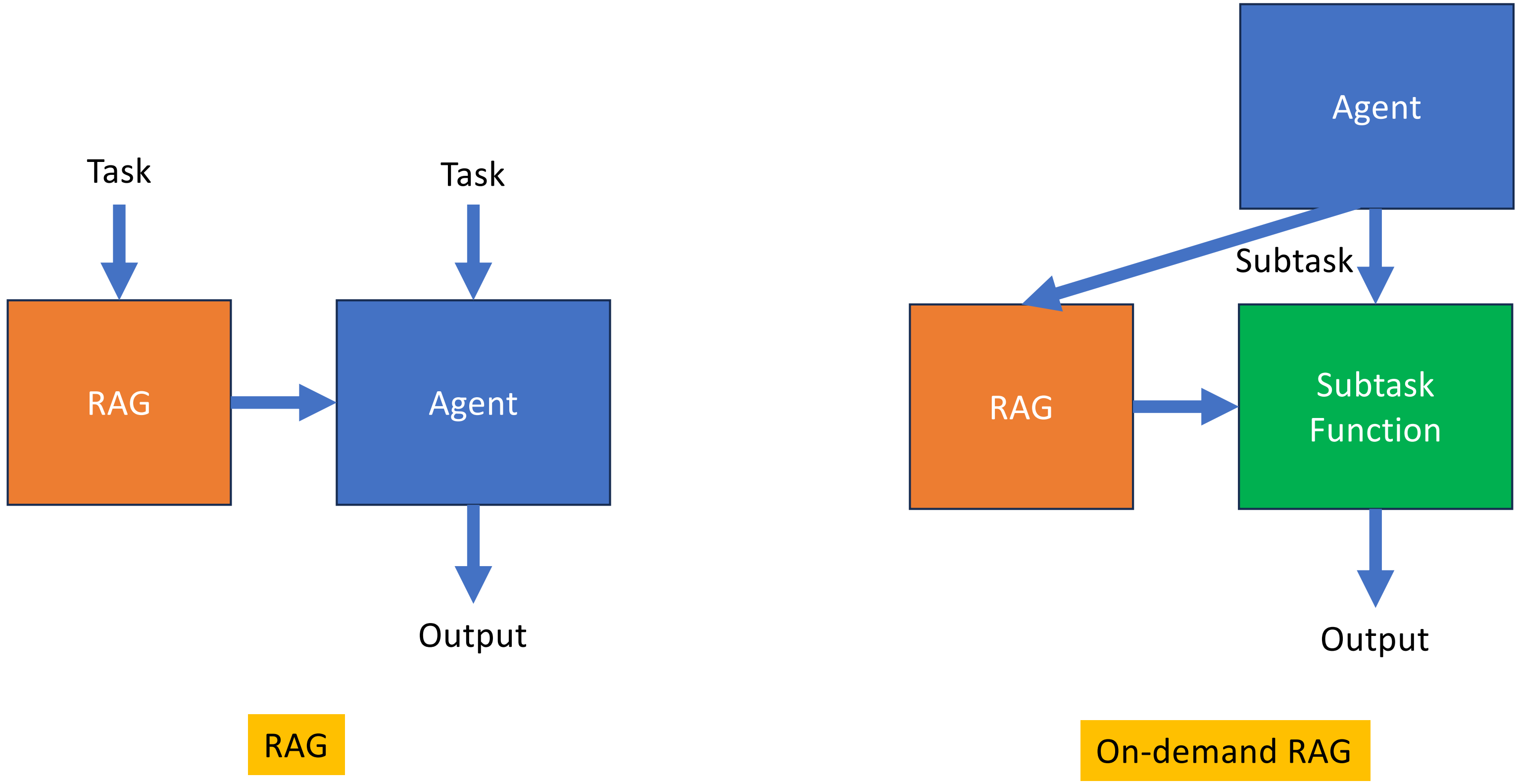
Retrieval Augmented Generation

Where to put it

RAG

- Retrieval Augmented Generation can help by putting several in-context examples in the prompt for better generation
- But placing it at the start of main prompt means that it might interfere with every process even when unrelated
- Should do on-demand RAG

RAG vs On-demand RAG



A Better RAG System

- Consider also using RAG based off LLM-prompts (if memory is small)
- Consider RAG with multiple abstraction spaces
- Consider doing RAG based off entity filtering first
- Consider Graph-based RAG (e.g. Knowledge Graphs, GraphRAG)

Memory

Memory can aid, memory can also harm

Learning from Previous Tasks

- Learning from previous tasks can help with better performance in current task, **if the task is similar**
- Otherwise, it can hinder learning
- Insight: May need two separate agents, one with memory and one without

Questions to Ponder

- When should Agents be used end-to-end, as opposed to just using LLMs for each part of the process?

- How many tools should one Agent use to be reliable?

- How to improve RAG?